

In re Application of: Chkodrov et al.
Application No.: 09/539,090

Remarks

In the application, claims 1 through 7, 16, 17, and 22 through 24 are pending. No claims currently stand allowed.

The Final Office Action dated May 17, 2004, has been carefully considered. The Final Office Action rejects claims 1 through 10 and 12 through 24 under 35 U.S.C. § 102(b) as anticipated by U.S. Patent 5,790,861 ("Rose"). Claim 11 is rejected under 35 U.S.C. § 103(a) as obvious in light of Rose and U.S. Patent 6,385,769 ("Lewallen").

The present application and Rose both describe techniques for replacing a base software object class with another object class. However, the timing of this replacement differs, and that difference significantly affects the utility of these techniques.

Rose describes traditional base-class inheritance where the replacement *always occurs at compile or link time*. Applicants respectfully disagree that the portion of Rose cited by the Final Office Action shows replacement occurring *during program execution*. In Rose's Figure 2, the base and replacement objects are compiled (step 210) and then linked (step 260). Rose's execution occurs in step 280 ("run") and *that step ends before class replacement can occur* (by looping back to steps 210, 220, and 260). Thus, in Rose the class replacement occurs *between* a first program execution and a second execution and *not during* program execution. The entire text of Rose's specification supports this view. For specific statements, see, e.g., the Abstract and column 3, lines 43 through 48 ("As a result, even if header file class definitions are subsequently modified, it will often be possible to generate updated executable code *by recompiling the modified header files and relinking the resulting object code*, without necessarily recompiling the application program code.") (emphasis added).

In contrast to Rose's compile- (or link-) time replacement, the present invention allows class replacement *during program execution*. This is true polymorphism and even works for a replacement class *that was unknown at compile time*, a feature clearly not disclosed by Rose. The present specification emphasizes this timing on page 14, lines 8 through 10 ("In contrast [with a Rose-type system], in the example given in FIG. 2, the replacement class B *may be designed after the reusable module 70 was generated*."), page 14, lines 14 through 17 ("The class replacement in accordance with the invention enables *dynamic creation of objects of a new derived class* instead of objects of a base class *during execution of the existing code in the program*."), page 15, line 24,

In re Application of: Chkodrov et al.
Application No.: 09/539,090

through page 16, line 1 (“*During the execution of the program 94*, the new code in the module 90 calls special instructions to indicate, or register, the class replacement relationship between the classes A and B in a particular context.”), page 30, line 24 through 27 (“In contrast, the automatic class replacement is carried out by *dynamically* passing the correct CLSREF value for the class of the object to be created.”), and page 38, line 38, through page 39, line 6 (“Although in this example the Create function is invoked in the ‘main’ portion of the program, it will be appreciated that *the instruction to create objects of the replaceable classes may be included in an existing reusable module that was developed before the replacement classes were designed*. Note that the old code needs to know only about the CobjCreator object and the replaceable base classes. *It did not have to know about the new derived classes at compile time* in order to be reusable for creating objects of the new classes.”) (emphasis added).

To further emphasize this timing distinction over Rose, all pending independent claims (1, 16, and 22) are amended. For example, claim 1 now clarifies “during program execution” as follows:

Claim 1. instructing in the second source file to replace the base class with the replacement class to cause creation of an object of the replacement class when the instruction in the first module to create an object of the base class *is executed during program execution, the program execution occurring after the compiling of the first source file, the creation of the object of the replacement class occurring without recompiling the first source file;*

(Emphasis added.) (Claims 16 and 22 have similar language.)

Rose simply does not teach this execution-time class replacement. As Rose neither anticipates nor renders obvious this feature of every pending claim, applicants request that the rejections be withdrawn and that all currently pending claims be allowed.

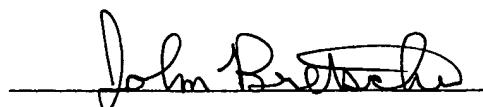
Conclusion

The application is considered in good and proper form for allowance, and the Examiner is respectfully requested to pass this application to issue. If, in the opinion of the Examiner, a

In re Application of: Chkodrov et al.
Application No.: 09/539,090

telephone conference would expedite the prosecution of the subject application, the Examiner is invited to call the undersigned attorney.

Respectfully submitted,

A handwritten signature in black ink, appearing to read "John T. Bretscher", is written over a horizontal line.

John T. Bretscher, Reg. No. 52,651
One of the Attorneys for Applicants
LEYDIG, VOIT & MAYER, LTD.
Two Prudential Plaza, Suite 4900
180 North Stetson
Chicago, Illinois 60601-6780
(312)616-5600 (telephone)
(312)616-5700 (facsimile)

Date: September 17, 2004